Multi-Robot Localization by Observation Merging

Ahmet Erdem and H. Levent Akın

Boğaziçi University, Department of Computer Engineering, 34342, Istanbul, Turkey

Abstract. In robot soccer, self-localization of robots may fail because of perception failure, falling down or by being pushed by another robot. In this study, our goal is to improve self localization of robots using the teammate robots' perceptions. Robots which have perceived more landmarks and have moved less can share their localization and observations with other robots to improve localization accuracy. Currently, in the RoboCup Standard Platform League it is not feasible to identify the jersey number of robots using vision. Therefore, we merge perceptions of all robots depending on their reliability, and then identify them in a probabilistic manner to increase localization performance and provide a common world model that can be used for planning. We show that our approach has a significant advantage with respect to single robot localization on estimating the poses of the robots.

Keywords: multiagent localization, map merging, Standard Platform League

1 Introduction

In the RoboCup Standard Platform League (SPL), robots use landmarks such as field lines and goal bars as observations for localization. However, these are symmetric with respect to the center line which makes localization harder. When a robot is near to the center of the field, its observations may be due to observations from either side of the field causing localization errors. The cases where the robots are kidnapped, even though not very frequent, also exist. When robots interfere with each others locomotion, this also results in serious localization errors. In addition, noise in perception and motion make it an even harder problem. In order to eliminate such noise and solve the kidnapping problem, we need our robots to work collaboratively. Robots merging their perceptions in a consistent way in order to reach a common world perception leading to better localization is a viable approach.

Collaborative localization methods can be implemented for either stationary or mobile robots. Localization of stationary robots are also known as *network localization* [1, 2]. When it comes to mobile robots, most previous works assume that the identity of the robots can be perceived by the individual robots. But in SPL this is not the case. A similar problem with ours is solved in [3] where they studied absolute mutual localization with anonymous relative position measurements, with perception modules that do not provide identification of the robots. When designing a multirobot algorithm, we also need to consider time constraints, for during the game, the robots have to respond to changes quickly.

In this study, we introduce an efficient novel collaborative localization method without identification for robot soccer. The problem we address is to provide better localization for Nao [4] robots on a soccer field during an SPL game by merging their perceptions in a consistent way. Each robot is able to self-localize, and in our study we use a Monte Carlo Localization (MCL) algorithm [5] with a set of extensions as described in [6]. In MCL unique landmarks in the field are essential, however, there are no unique landmarks on the SPL field except the center circle. For this reason, we aim to use the team members as unique landmarks. There are, however, two major problems:

- There is no identification. In other words, the robots can perceive that there is a team mate in its field of view, but cannot perceive its jersey number. So, the robots can only observe that at position (x, y) there is a teammate instead of that player 3 is at position (x, y).
- Self-localization is noisy, because robots perceptions and motions are noisy. Therefore, we cannot rely on a single robot. If its localization knowledge is erroneous, the position of a robot seen by it is also wrong since perceptions are relative. For this reason, we need to estimate reliabilities of the robots before processing the information they send. In addition, a robot with good position estimate but bad orientation estimate may cause wrong position estimates of visible teammates.

In [7], a solution to this problem is proposed. They introduced a collaborative multi-robot localization (MRL) approach which improves performance of the self-localization. They make use of relative orientations of the robots. In this paper we extend their approach so that we make use of not only their relative orientations but also the relative distances. Another advantage of our approach is that it generates a common world model for the robots, which may be useful for solving planning problems.

The organization of the rest of the paper is as follows. In Section 2 we present the proposed approach. The experiments and the results are given in Section 3 and the Conclusions in Section 4.

2 Proposed Approach

In collaborative multi-robot localization the robots share some information that is used in the localization process. We describe below the details of the communication between the robots and the developed multi-robot localization approach.

2.1 Messaging

The Nao robots can communicate with each other via a wireless network. They send and receive broadcast messages including some important data such as the position of the ball. In order to realize our algorithm, we add a new message component which includes the following information:

- player number of the sender,
- orientation of the sender,
- mean and variance values of the sender's estimated position,
- number of robots seen by the sender and their relative positions.

2.2 Multi-robot Localization

The Circle Method. We represent the possible locations of the robots by circles with orientations. We use circles, because when a robot is observed, its orientation is unknown. If we consider a robot as a thick line, a known center and all possible orientations construct a circle. Each circle can be constructed by the coordinates of its center, an orientation, a reliability value between 0 and 1, and its type. Since they represent the robots, the circles can perceive each other. There are three types of circles:

- *A* Circle: This is the circle which a robot claims that it is on. Initially, the number of *A* circles must be equal to the number of robots.
- *B* Circle: This is the circle which a robot claims that it perceives a teammate on.
- *O* Circle: This is the circle which is occupied by a robot as its correct location. Initially, there is no *O* circle. The number of *O* circles eventually increases while the number of *A* and *B* circles decreases.

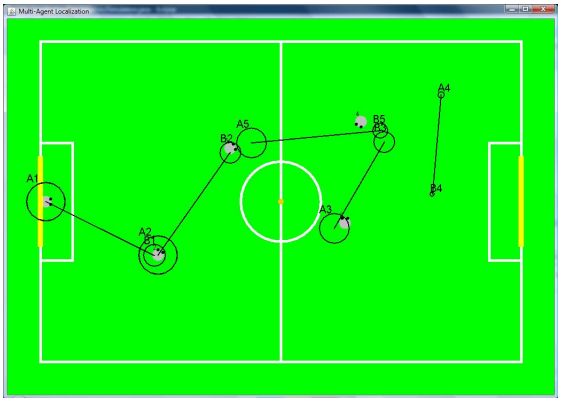


Fig. 1. Initial Circle Construction

We assume that particles in the self-localization process (MCL) have a Gaussian distribution. We can calculate the variance and mean of the particles. Robots consider these means as their estimated positions. For an *A* circle, there is a negative correlation between the variance σ_A^2 and reliability ρ_A. We can say that over a certain variance threshold $maxVariance$, the robots are completely lost and have 0 percent reliability as in Equation 1:

$$\rho_A = \max\left(0, 1 - \frac{\sigma_A^2}{maxVariance}\right) \tag{1}$$

The radius of a *B* circle is related to the distance between its location and its perceiver robots location. As the distance increases, the reliability decreases. Distance is normal-

ized by the $visionDistanceLimit$ which is the maximum distance for perceiving a teammate visually as given in Equation 2:

$$\rho_B = \rho_A \cdot \left(1 - \left(\frac{\text{distance}_{A,B}}{visionDistanceLimit}\right)^2\right) \tag{2}$$

The radius of a circle is proportional to the reliability value ρ. So, robots with low variance form larger *A* and *B* circles. In this way, the intersection possibility of reliable data increases. Initially, we construct *A* circles for each robots self location estimate and *B* circles for their perceived teammates as shown in Figure 1.

Map Merging. After the construction of the circles, we may notice that some circles overlap as seen Figure in 1. We have to merge them in order to have one consistent world model. Binary merging of circles continues until there are no intersecting circles. In order to merge two circles, the following conditions have to be satisfied:

- They have to be perceived by different robots. This ensures that we do not see two robots, which are close, as a single robot.
- At most one of them may be an *A* circle. This ensures that we do not merge two different robot positions as one.

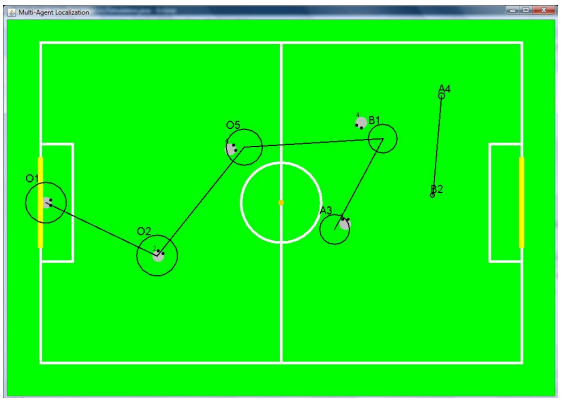


Fig. 2. Map Merging

The Merging Process: The merging process goes through the following steps:

1. The new center is the center of mass of the two circles. The coordinates of the center is given in Equation 3.

$$\begin{aligned} x_{\text{merged}} &= \frac{(x_{c1} \cdot \text{area}_{c1} + x_{c2} \cdot \text{area}_{c2})}{\text{area}_{c1} + \text{area}_{c2}} \\ y_{\text{merged}} &= \frac{(y_{c1} \cdot \text{area}_{c1} + y_{c2} \cdot \text{area}_{c2})}{\text{area}_{c1} + \text{area}_{c2}} \end{aligned} \tag{3}$$

2. New reliability is calculated as in Equation 4. It is the probability that at least one of the circles is in the correct position. Therefore, the merged circle is more reliable than its constructor circles.

$$\rho_{\text{merged}} = 1 - (1 - \rho_{c1}) \cdot (1 - \rho_{c2}) \tag{4}$$

3. *A* circles which perceive these two circles start to perceive the new one instead of them.
4. If one of the merged circles is an *A* circle, the new circle perceives the circles which were perceived by the *A* circle before.
5. Constructor circles are deleted.
6. Finally, reliabilities of the circles which perceive a merged circle increase such that they have reliabilities greater than or equal to the merged circle's reliability. Reliabilities of the other *B* circles which do not participate in merging also increase when the reliabilities of *A* circles which perceive them increase. Consensuses between robots lead to more reliable robots.

After all intersected circles are merged; *A* circles with higher reliability than a certain threshold are converted to *O* circles as seen in Figure 2. Map merging algorithm is given in Algorithm 1.

Algorithm 1 Map Merging

```
1: procedure MERGEMAP(circles)
2:     while haveIntersections(circles) do
3:         mergeCircles(pickAnIntersectedPair(circles))
4:     end while
5:     for all c_i ∈ circles do
6:         if c_i.type = a and c_i.reliability > reliabilityThreshold then
7:             c_i.type ←o
8:         end if
9:     end for
10: end procedure
```

Occupying Circles Based on the Bounty System. We want our players to occupy a circle which suits best for them. The best is selected based on the following three criteria. For each player which has not occupied any circle yet, we give bounty points to possible locations it can occupy. These locations are the *A* circle itself and *B* circles which are not perceived by this circle. Pairs $(\text{player}_i, \text{circle}_j)$ gain bounty points based on the reliability of the circle. We developed following metrics to assess reliability of a circle:

– *Bounty by Distance:* Pairs gain bounty points related to the distance between circle_i and circle_j. Circles which are close to the current *A* circle gain more bounty points. In other words, there is a negative correlation between the distance to the *A* circle

and the bounty points. This rewards the circles which are near to the location estimated by the robot itself. The *A* circle centered at the estimated location of the robot gets the highest bounty as in Equation 5.

$$\text{bounty}_{i,j} = \text{bounty}_{i,j} + \text{BOUNTY}_{dist} \cdot \left(1 - \frac{\text{dist}(c_i, c_j)}{\text{normalizer}}\right) \tag{5}$$

– *Bounty by Reliability:* Circles gain bounty points with respect to their reliability points. There is a positive correlation between the bounty points and reliabilities as in Equation 6

$$\text{bounty}_{i,j} = \text{bounty}_{i,j} + \text{BOUNTY}_{reliability} \cdot \text{reliability}_j \tag{6}$$

– *Bounty by Visibility:* We check candidate circles whether they are in the line-of-sight of occupied circles and they are not perceived by them. If this is the case, the circle loses bounty points. The magnitude of the loss depends on the visibility distribution which is calculated by Equation 7. If the circle is in the line-of-sight of more than one occupied circle, losses are summed. (Actually, all circles gain visibility points, no one loses but they lose relative to the others' gains.) The visibility bounty point is calculated by Equation 8.

$$\text{visibility}(k, j) = \left(1 - \frac{\text{dist}(c_k, c_j)}{visionDistanceLimit}\right) \cdot \left(1 - \frac{|c_k.headOrientation - \text{slope}(c_k, c_j)|}{visionDegreeLimit}\right) \tag{7}$$

$$\text{bounty}_{i,j} = \text{bounty}_{i,j} + \text{BOUNTY}_{visibility} \cdot \left(1 - \sum_{k \in O} \text{visibility}(k, j)\right) \tag{8}$$

After these operations, the pair with the biggest bounty is found. (This bounty should be above a certain limit.) If the circle is one of the *B* circles, the *A* circle which represents this player moves to this *B* circle. All the circles perceived by the *A* circle also move. The circle is converted to an *O* circle for this player. The process given in Algorithm 2 repeats until it cannot generate an *O* circle.

Algorithm 2 Occupying Circles Based On Bounty System

```
1: procedure FINDHIGHESTBOUNTYANDOCCUPY(c1, c2)
2:     giveBountyByReliability(bounties)
3:     giveBountyByVisibility(bounties)
4:     giveBountyByDistance(bounties)
5:     [i, j] ←max(bounties)
6:     occupy(i, j)
7: end procedure
```

By this method, the robots select the best circles for themselves whenever they can. Being close to the robot's estimated position, high possibility of having a robot on it

and low possibility of visibility contradictions makes a circle the best choice as seen in Figure 3. One can tune the bounty coefficients to have better results. For example; if the bounty coefficient of distance is too small compared to others, it may cause "teleportation" situations.

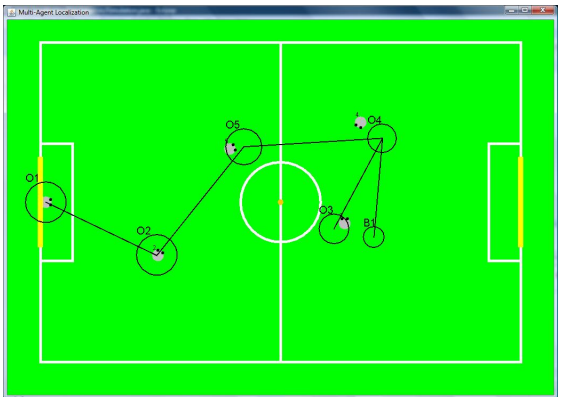


Fig. 3. Occupying Circles Based on Bounty System

Rotation Correction. After these procedures, if there are some robots which perceive at least one B circle which is not perceived by the others, we can say that this robot may have incorrect knowledge of its orientation. Then we try to rotate it between $-\theta$ and $+\theta$ degrees. The orientation which causes the maximum number of intersections for the circles perceived by the robot becomes the suggested orientation for the robot.

After rotation correction, circles are again merged until there is no intersection as seen in Figure 4. If there are still B circles which are perceived by only one O circle, this O circle is converted back to an A circle in order to prevent misleading localization information, because these circles may have true positions but faulty orientations.

Combining with Monte Carlo Localization. If a robot can occupy a circle at the end of the Multi-Robot Localization process, this circle is used in self-localization of the robot. Since the self-localization is a Monte Carlo Localization, we inject some particles around the center of the circle. These particles are constructed by adding 2D Quasi-Gaussian noise to the center. Quasi-Gaussian particles are preferred to Gaussian ones, because a previous work [8] has shown that quasi-random numbers have great advantage over actual random numbers in localization. Figure 5 illustrates the difference between Gaussian and Quasi-Gaussian distributions used in this study.

These particles are injected just after the resampling process. The number of injected particles is proportional to the reliability (i.e. radius) of the circle. Orientation of each injected particle is a Gaussian random number with the circle's orientation as a

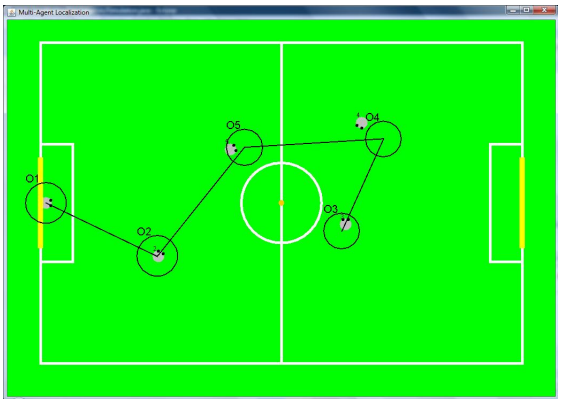


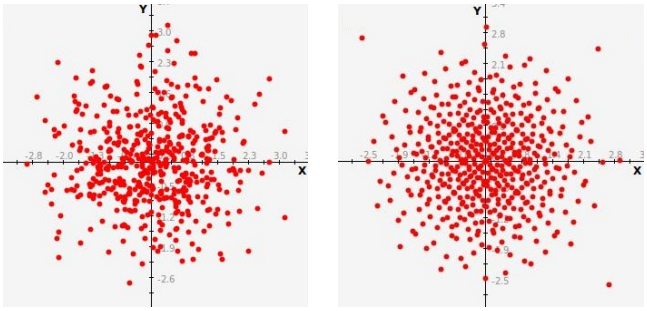
Fig. 4. Rotation Correction

Algorithm 3 Rotation Correction

procedure CORRECTROTATION($circles$)
  for all $c_i \in circles$ **do**
    if $c_i.type = o$ **then**
      $findAndSetBestRotation(c_i)$
    end if
  end for
  $mergeMap(circles)$
  for all $c_i \in circles$ **do**
    if $c_i.type = b$ (and) $sizeOf(c_i.perceivedByCircles) = 1$ **then**
      $c_i.perceivedByCircles[0].type \leftarrow$a
    end if
  end for
end procedure

Algorithm 4 Find And Set Best Rotation

procedure FINDANDSETBESTROTATION(c)
  $maxNumOfIntersections \leftarrow 0$
  for $i = -\theta$ **to** θ **do**
    $numOfIntersections \leftarrow$c.rotateTo(c.orientation + i)
    if $numOfIntersections >$maxNumOfIntersections **then**
      $maxNumOfIntersections \leftarrow$numOfIntersections
      $bestRotation \leftarrow$c.orientation + i
    end if
  end for
  $c.rotateTo(c.orientation + bestRotation)$
end procedure



(a) 2D Gaussian particles. (b) 2D Quasi-Gaussian Particles

Fig. 5. The Difference Between Gaussian and Quasi-Gaussian Random Numbers.

mean. Since we claim that we inject particles to the real position with the real orientation, these particles gain weights by observations. After resampling, more particles will be sampled from these particles due to their large weights. By this method, we expect to see particles converging to the real position of the robot after a few steps.

3 Experiments and Results

3.1 Simulation Experiments

In order to make the initial tests of the proposed approach, we have developed a 2D robot simulator since we want to control motion and perception noises.The other reason is that we can make a large number of experiments in a small amount of time without harming the actual robots. We can monitor the robot's real positions and estimated positions via the 2D simulation interface. The robots walk from one point to another point. We add noise to both motion and perception of the robot. Noise is proportional to the magnitude of actions. If a robot walks a large distance or perceives another robot which is far away, the noise gets larger.

We first tested the performance of our method with five robots using our simulator. In each simulation, we allow the algorithm to run for 100 steps. Different polygonal paths are planned for each robot and the robots try to follow these paths with respect to their estimated positions. In each step, the robots travel at most a 32 cm length path and rotate when necessary. Since the localization is noisy, the robots cannot follow the determined paths exactly. How much they deviate from the actual path depends on the quality of the localization process.

We have tested the algorithm with 1000 different runs and the average position error per player per step and average orientation error per player per step have been calculated as the measurement of performance. Setups with different number of stationary robots (robots which do not move and observe others) have also been tested. We have found that our method improves localization significantly in terms of position estimates. From Table 1 it can be observed that the performance improvement increases as the number of

stationary robots increases. The improvement in orientation is not significant as shown in Table 2.

Table 1. Position Error in the 2D Simulation Environment

Number of Stationary Robots	Error Without MRL (mm)	Error With MRL (mm)	Improvement Percentage
1	402	370	8.0
2	353	248	29.7
3	254	178	29.9
4	125	61	51.2

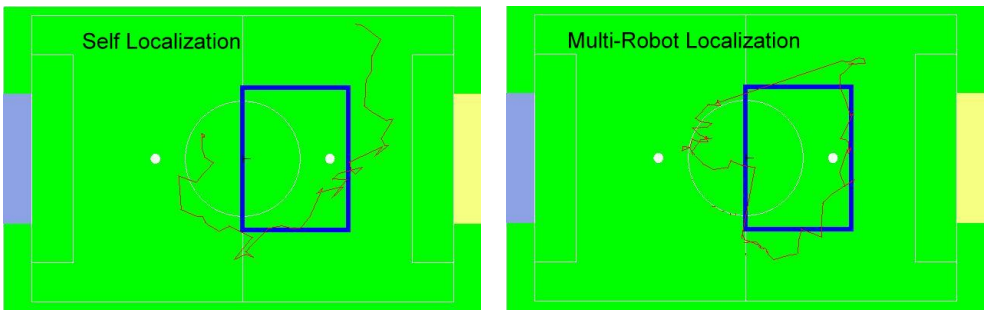
Table 2. Orientation Error in the 2D Simulation Environment

Number of Stationary Robots	Error Without MRL (degree)	Error With MRL (degree)	Improvement Percentage
1	35	34	0.03
2	24	21	0.13
3	19	19	0.00
4	19	17	0.11

3.2 Experiments with Real Robots

After the initial simulator tests we tested our algorithm on Nao robots in the SPL field. In order to get the real position of the robot on the field we have modified the robot tracking system previously developed by Kavaklıoğlu [9] in his MSc thesis. In this setup, there are four webcams fitted to the ceiling of the field. They are calibrated for giving exact positions. We put a marker on top of the robot which we want to track. The marker consists of one blue and two brown blobs. These colors dont belong to any other objects. The blue one is for finding the orientation of the robot. The robot tracking system can easily detect these markers and send their positions on network. We receive these messages in order to draw the current position and the followed path and to calculate the localization errors.

We have designed a rectangular path for the localization test. We have tracked the position of the robot during the test by the cameras. MRL was disabled for the control group. In most of the cases in which MRL was disabled, robots left the field whenever they could not observe any landmarks. When we enabled MRL, robots completed the path with less localization errors. We show a sample from our experiments in Figure 6.



(a) Without MRL (b) With MRL

Fig. 6. Effect of MRL on path following (Real Robots)

4 Conclusions

Collaborative localization is a challenging problem in multi-robot systems, when used efficiently it can lead to an overall improvement in localization. In the SPL, localization is a major issue since over the years unique landmarks have been removed one by one. In this study we introduce a novel method for MRL without player identification. We not only aim to improve self-localization performance of the robots, but also aim to provide a common world model for them to be used in planning. Since robot soccer is a real-time event, we have considered time constraints and proposed a time-efficient method.

The simulation experiments have shown that our approach has a significant advantage over single robot localization on estimating the position and the orientation, but we could not observe the same performance when we did the real world experiments. The main reason is that the real world may have more noise than we expected. In addition, we need better perception and motion modules in order to decrease these noises.

We have to mention that in the cases where the lost robot cannot see its teammates, we cannot give orientation information for that robot. It is a shortcoming of our method. It can be eliminated, if the ball is visible to that robot and at least one reliable robot. The position given by MRL and relative distance and orientation to the ball are sufficient for estimating the orientation.

The most challenging part of our method is determining which robots are reliable. Our criterion is the variance of particles in MCL, but sometimes particles may converge to a wrong position and the robots may not be aware that they are lost. As a further work, we intend to work on better estimating the reliability.

Acknowledgments

This work is supported by Boğaziçi University Research Fund through project 13A01P3.

References

[1] J. Aspnes, T. Eren, D.K. Goldenberg, A.S. Morse, W. Whiteley, Y.R. Yang, B. D O Anderson, and P.N. Belhumeur. A theory of network localization. *IEEE Transactions on Mobile Computing*, 5(12):1663–1678, Dec 2006.

[2] R. Aragues, L. Carlone, G. Calafiore, and C. Sagues. Multi-agent localization from noisy relative pose measurements. In *2011 IEEE International Conference on-Robotics and Automation (ICRA 2011)*, pages 364–369, May 2011.

[3] A. Franchi, G. Oriolo, and P. Stegagno. Probabilistic mutual localization in multi-agent systems from anonymous position measures. In *49th IEEE Conference on Decision and Control (CDC 2010)*, pages 6534–6540. IEEE, 2010.

[4] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier. The nao humanoid: a combination of performance and affordability. *CoRR abs/0807.3223*, 2008.

[5] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.

[6] K. Kaplan, B. Çelik, T. Meriçli, Ç. Mericli, and H. L. Akın. Practical extensions to vision-based monte carlo localization methods for robot soccer domain. In A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, editors, *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020, pages 420–427, 2006.

[7] N. E. Ozkucur, B. Kurt, and H. L. Akın. A collaborative multi-robot localization method without robot identification. In *RoboCup 2008: Robot Soccer World Cup XII*, volume 5399, pages 189–199. Springer, 2009.

[8] Bingbing Liu, Xi Zheng, Xiaojun Wu, and Yiguang Liu. Quasi monte carlo localization for mobile robots. In *12th International Conference on Control Automation Robotics & Vision (ICARCV 2012)*, pages 620–625, 2012.

[9] C. Kavaklıoğlu. Developing a probabilistic post-perception module for mobile robotics. Master's thesis, Boğaziçi University, 2009.